UNITED STATES PATENT APPLICATION FOR:

CONTENT MINING FOR VIRTUAL CONTENT REPOSITORIES

Inventors:

Gregory Smith
James Owen

CERTIFICATE OF MAILING BY "EXPRESS MAIL"
UNDER 37 C.F.R. §1.10

"Express Mail" mailing label number: EV327618617US
Date of Mailing: 2/5/04

I hereby certify that this correspondence is being deposited with the United States Postal Service, utilizing the "Express Mail Post Office to Addressee" service addressed to: **MAIL STOP PATENT APPLICATION, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450** and mailed on the above Date of Mailing with the above "Express Mail" mailing label number.

Tina M. Galdos (Signature)
Name: Tina M. Galdos
Signature Date: 2/5/04

CONTENT MINING FOR VIRTUAL CONTENT REPOSITORIES

Inventors:

Gregory Smith
James Owen

COPYRIGHT NOTICE

CLAIM OF PRIORITY

[0002] This application claims priority from the following application, which is hereby incorporated by reference in its entirety:

[0003] SYSTEM AND METHOD FOR A VIRTUAL CONTENT REPOSITORY, U.S. Provisional Patent Application No. 60/449,154, Inventors: James Owen, et al., filed on February 20, 2003. (Attorney's Docket No. BEAS-1360US0)

[0004] SYSTEMS AND METHODS FOR PORTAL AND WEB SERVER ADMINISTRATION, U.S. Provisional Patent Application No. 60/451,174, Inventors: Christopher Bales, et al., filed on February 28, 2003. (Attorney's Docket No. BEAS-1371US0)

CROSS-REFERENCE TO RELATED APPLICATIONS

[0005] This application is related to the following co-pending applications which are each hereby incorporated by reference in their entirety:

[0006] VIRTUAL REPOSITORY CONTENT MODEL, U.S. Application No. 10/618,519, Inventors: James Owen, et al., filed on July 11, 2003. (Attorney's Docket No. BEAS-1361US0)

[0007] VIRTUAL REPOSITORY COMPLEX CONTENT MODEL, U.S. Application No. 10/618,380, Inventors: James Owen, et al., filed on July 11, 2003. (Attorney's Docket No. BEAS-1364US0)

[0008] SYSTEM AND METHOD FOR A VIRTUAL CONTENT

REPOSITORY, U.S. Application No. 10/618,495, Inventors: James Owen, et al., filed on July 11, 2003. (Attorney's Docket No. BEAS-1363US0)

[0009] VIRTUAL CONTENT REPOSITORY APPLICATION PROGRAM INTERFACE, U.S. Application No. 10/618,494, Inventors: James Owen, et al., filed on July 11, 2003. (Attorney's Docket No. BEAS-1370US0)

[0010] SYSTEM AND METHOD FOR SEARCHING A VIRTUAL REPOSITORY CONTENT, U.S. Application No. 10/619,165, Inventor: Gregory Smith, filed on July 11, 2003. (Attorney's Docket No. BEAS-1365US0)

[0011] VIRTUAL CONTENT REPOSITORY BROWSER, U.S. Application No. 10/618,379, Inventors: Jalpesh Patadia et al., filed on July 11, 2003. (Attorney's Docket No. BEAS-1362US0)

[0012] FEDERATED MANAGEMENT OF CONTENT REPOSITORIES U.S. Application No. 10/618,513, Inventors: James Owen et al., filed on July 11, 2003. (Attorney's Docket No. BEAS-1360US1)

FIELD OF THE DISCLOSURE

[0013] The present invention disclosure relates generally to content management.

BACKGROUND

[0014] Content repositories manage and provide access to large data stores such as a newspaper archives, advertisements, inventories, image collections, etc. A content repository can be a key component of a Web application such as a Web portal, which must quickly serve up different types of content in response to a particular user's requests. However, difficulties can arise when trying to integrate more than one vendor's content repository. Each may have its own proprietary application program interface (API), conventions for manipulating content, and data formats. Performing a search across different repositories, for example, could require using completely different search mechanisms and converting each repository's search results into a common format. Furthermore, each time a repository is added to an application, the application software must be modified to accommodate these differences.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] **Figure 1** is an illustration of a virtual content management framework in one embodiment of the invention.

[0016] **Figure 2** is an illustration of functional layers in one embodiment of the invention.

[0017] **Figure 3** is an illustration of objects used in connecting a repository to a virtual content repository in one embodiment of the invention.

[0018] **Figure 4** is an exemplary content model in one embodiment of the invention.

[0019] **Figure 5** is an exemplary service model in one embodiment of the invention.

[0020] **Figure 6** is an illustration of *NopeOps* service interaction in one embodiment of the invention.

[0021] **Figure 7** is an illustration of a virtual content repository browser in one embodiment of the invention.

[0022] **Figure 8** is an illustration of a content editor in one embodiment of the invention.

[0023] **Figure 9** is an illustration of a schema editor in one embodiment of the invention.

[0024] **Figure 10** is an illustration of a property editor in one embodiment of the invention.

[0025] **Figure 11** is an illustration of content mining system in one embodiment of the invention.

[0026] **Figure 12** is an illustration of a content mining process in one embodiment of the invention.

DETAILED DESCRIPTION

[0027] The invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that references to "an" or "one" embodiment in this disclosure are not necessarily to the same embodiment, and such references mean at least one.

[0028] In the following description, various aspects of the present invention will be described. However, it will be apparent to those skilled in the art that the

present invention may be practiced with only some or all aspects of the present invention. For purposes of explanation, specific numbers, materials and configurations are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well-known features are omitted or simplified in order not to obscure the present invention.

[0029] Parts of the description will be presented in data processing terms, such as data, selection, retrieval, generation, and so forth, consistent with the manner commonly employed by those skilled in the art to convey the substance of their work to others skilled in the art. As well understood by those skilled in the art, these quantities take the form of electrical, magnetic, or optical signals capable of being stored, transferred, combined, and otherwise manipulated through electrical and/or optical components of a processor and its subsystems.

[0030] Various operations will be described as multiple discrete steps in turn, in a manner that is most helpful in understanding the present invention, however, the order of description should not be construed as to imply that these operations are necessarily order dependent. In particular, these operations need not be performed in the order of presentation.

[0031] Various embodiments will be illustrated in terms of exemplary classes and/or objects in an object-oriented programming paradigm. It will be apparent to one skilled in the art that the present invention can be practiced using any number of different classes/objects, not merely those included here for illustrative purposes. Furthermore, it will also be apparent that the present invention is not limited to any particular software programming language or programming paradigm.

[0032] **Figure 1** is an illustration of a virtual content management framework in one embodiment of the invention. A content repository **108** is a searchable data store. Such systems can relate structured content and unstructured content (e.g., digitally scanned paper documents, eXtensible Markup Language, Portable Document Format, Hypertext Markup Language, electronic mail, images, video and audio streams, raw binary data, etc.) into a searchable corpus. Content repositories can be coupled to or integrated with content management systems. Content management systems provide for content life cycle management (e.g. versioning), content review and approval, automatic content classification, event-driven content processing,

process tracking and content delivery to other systems. For example, if a user fills out a loan application on a web portal, the web portal can forward the application to a content repository which, in turn, can contact a bank system, receive notification of loan approval, update the loan application in the repository and notify the user by rendering the approval information in a format appropriate for the web portal.

[0033] A virtual or federated content repository (hereinafter referred to as "VCR") **100** is a logical representation of one or more individual content repositories **108** such that they appear and behave as a single content repository from an application program's **110** standpoint. This is accomplished in part by use of an API (application program interface) **104** and an SPI (service provider interface) **102**. An API describes how an application program, library or process can interface with some program logic or functionality. By way of a non-limiting illustration, a process can include a thread, a server, a servlet, a portlet, a distributed object, a web browser, or a lightweight process. An SPI describes how a service provider (e.g., a content repository) can be integrated into a system of some kind. SPI's are typically specified as a collection of classes/interfaces, data structures and functions that work together to provided a programmatic means through which a service can be accessed and utilized. By way of a non-limiting example, APIs and SPIs can be specified in an object-oriented programming language, such as Java™ (available from Sun Microsystems, Inc. of Mountain View, California) and C# (available from Microsoft Corp. of Redmond, Washington). The API and SPI can be exposed in a number of ways, including but not limited to static libraries, dynamic link libraries, distributed objects, servers, class/interface instances, etc.

[0034] In one embodiment, the API presents a unified view of all repositories to application programs and enables them to navigate, perform CRUD (create, read, update, and delete) operations, and search across multiple content repositories as though they were a single repository. Content repositories that implement the SPI can "plug into" the VCR. The SPI includes a set of interfaces and services that repositories can implement and extend including schema management, hierarchy operations and CRUD operations. The API and SPI share a content model **106** that represents the combined content of all repositories **108** as a hierarchical namespace of nodes (or hierarchy). Given a node *N*, nodes that are hierarchically inferior to *N* are referred to as children of *N* whereas nodes that are hierarchically superior to *N* are referred to as parents of *N*. The top-most level of the hierarchy is called the federated

root. There is no limit to the depth of the hierarchy.

[0035] In one embodiment, content repositories can be children of the federated root. Each content repository can have child nodes. Nodes can represent hierarchy information or content. Hierarchy nodes serve as a container for other nodes in the hierarchy akin to a file subdirectory in a hierarchical file system. Content nodes can have properties. In one embodiment, a property associates a name with a value of some kind. By way of a non-limiting illustration, a value can be a text string, a number, an image, an audio/visual presentation, binary data, etc. Either type of node can have a schema associated with it. A schema describes the data type of one or more of a node's properties.

[0036] **Figure 2** is an illustration of functional layers in one embodiment of the invention. API **200** is layered on top of SPI **202**. The SPI layer isolates direct interaction with repositories **212** from the API. In one embodiment, this can be accomplished at run-time wherein the API library dynamically links to or loads the SPI library. In another embodiment, the SPI can be part of a server process such that the API and the SPI can communicate over a network. The SPI can communicate with the repositories using any number of means including, but not limited to, shared memory, remote procedure calls and/or via one or more intermediate server processes.

[0037] Referring again to **Figure 2** and by way of a non-limiting example, content mining facilities **204**, portlets **206**, tag libraries **208**, applications **210**, and other libraries **218** can all utilize the API to interact with a VCR. Content mining facilities can include services for automatically extracting content from the VCR based on parameters. Portlet and Java ServerPages™ tag libraries enable portals to interact with the VCR and surface its content on web pages. (Java ServerPages is available from Sun Microsystems, Inc.) In addition, application programs and other libraries can be built on top of the API.

[0038] In one embodiment, the API can include optimizations to improve the performance of interacting with the VCR. One or more content caches **216** can be used to buffer search results and recently accessed nodes. Content caches can include node caches and binary caches. A node cache can be used to provide fast access to recently accessed nodes. A binary cache can be used to provide fast access to the data associated with each node in a node cache. The API can also provide a configuration facility **214** to enable applications, tools and libraries to configure content caches and the VCR. In one embodiment, this facility can be implemented as a Java

Management Extension (available from Sun Microsystems, Inc.). Exemplary configuration parameters are provided in **Table 1**.

CONFIGURATION PARAMETERS
Active state for a binary cache of a repository (i.e., turn the cache on or off).
Maximum number of entries for a binary cache of a repository.
Time-to-live for entries in a binary cache of a repository.
Repository name.
Active state for a node cache of a repository (i.e., turn the cache on or off).
Max entries for a node cache of a repository.
Time-to-live for entries in a node cache of a repository.
Password and username for a repository.
Read-only attribute for the repository.
Class name of the repository implementation.
Additional key/value properties.

Table 1: Exemplary Configuration Parameters

[0039] **Figure 3** is an illustration of objects used in connecting a repository to a VCR in one embodiment of the invention. In one embodiment, objects implementing API interface *RepositoryManager* **302** can serve as an representation of a VCR from an application program's standpoint. A *RepositoryManager connect()* method attempts to connect all available repositories with a current user's credentials to the VCR. By way of a non-limiting example, credentials in one embodiment can based on the Java™ Authentication and Authorization Service (available from Sun Microsystems, Inc.). Those of skill in the art will recognize that many authorization schemes are possible without departing from the scope and spirit of the present embodiment. Each available content repository is represented by an SPI *Repository* object **306-310**. The *RepositoryManager* object invokes a *connect()* method on a set of *Repository* objects. In one embodiment, a *RepositorySession* object (not shown) can be instantiated for each content repository to which a connection is attempted. In

one embodiment, the *RepositoryManager connect()* method can return an array of the *RepositiorySessions* to the application program, one for each repository for which a connection was attempted. Any error in the connection procedure can be described by the *RepositorySession* object's state. In another embodiment, the *RepositoryManager connect()* method can connect to a specific repository using a current user's credentials and a given repository name. In one embodiment, the name of a repository can be a URI (uniform resource identifier).

[0040] **Figure 4** is an exemplary content model in one embodiment of the invention. The content model is shared between the API and the SPI. Each box in **Figure 2** represents a class or an interface. Hollow tipped arrows connecting boxes indicate inheritance relationships wherein the class/interface from which the arrows emanate inherit from the class/interface to which the arrows point. Solid tipped arrows indicate that the objects of the class/interface from which the arrows emanate can contain or have references (e.g., pointers or addresses) to objects of the class/interface to which the arrows point. In one embodiment, each object in a VCR has an identifier that uniquely identifies it. An identifier can be represented by an *ID* **400** (or id). An id can contain the name of a content repository and a unique id provided to it by the repository. In one embodiment, the id class/interface can be made available through a common super class/interface **414** that can provide services such as serialization, etc.

[0041] In one embodiment, content and hierarchy nodes can be represented by a *Node* **402** (or node). A node has a name, an id, and can also include a path that uniquely specifies an the node's location in the VCR hierarchy. By way of a non-limiting example, the path can be in a Unix-like directory path format such as '/a/b/c' where '/' is a federated root, 'a' is a repository, 'b' is a node in the 'a' repository, and 'c' is the node's name. The *Node* class provides methods by with a node's parent and children can be obtained. This is useful for applications and tools that need to traverse the VCR hierarchy (e.g., browsers). Nodes can be associated with zero or more *Property* **404** objects (or properties). A property can have a name and zero or more values **406**. In one embodiment, a property's name is unique relative to the node to which the property is associated. A *Value* **406** can represent any value, including but not limited to binary, Boolean, date/time, floating point, integer or string values. If a property has more than one value associated with it, it is referred to as "multi-valued".

[0042] A node's properties can be described by a schema. A schema can be referred to as "metadata" since it does not constitute the content (or "data") of the VCR per se. Schemas can be represented by an *ObjectClass* **408** object and zero or more *PropertyDefinition* **410** objects. An *ObjectClass* has a schema name that uniquely identifies it within a content repository. A node can refer to a schema using the *ObjectClass* name. In another embodiment, a content node can define its own schema by referencing an *ObjectClass* object directly. In one embodiment, there is one *PropertyDefinition* object for each of a node's associated *Property* objects. *PropertyDefinition* objects define the shape or type of properties. Schemas can be utilized by repositories and tools that operate on VCRs, such as hierarchical browsers. By way of a non-limiting example, a hierarchy node's schema could be used to provide information regarding its children or could be used to enforce a schema on them. By way of a further non-limiting example, a VCR browser could use a content node's schema in order to properly display the node's values.

[0043] In one embodiment, a *PropertyDefinition* can have a name and can describe a corresponding property's data type (e.g., binary, Boolean, string, double, calendar, long, reference to an external data source, etc.), whether it is required, whether it is read-only, whether it provides a default value, and whether it specifies a property choice type. A property choice can indicate if a property is a single unrestricted value, a single restricted value, a multiple unrestricted value, or a multiple restricted value. Properties that are single have only one value whereas properties that are multiple can have more than one value. If a property is restricted, its value(s) are chosen from a finite set of values. But if a property is unrestricted, any value(s) can be provided for it. *PropertyChoice* objects **412** can be associated with a *PropertyDefinition* object to define a set of value choices in the case where the *PropertyDefinition* is restricted. A choice can be designated as a default value, but only one choice can be a default for a given *PropertyDefinition*.

[0044] A *PropertyDefinition* object may also be designated as a primary property. By way of a non-limiting example, when a schema is associated with a node, the primary property of a node can be considered its default content. The *isPrimary()* method of the *PropertyDefinition* class returns true if a *PropertyDefinition* object is the primary *PropertyDefinition*. By way of a further non-limiting example, if a node contained a binary property to hold an image, it could also contain a second binary property to represent a thumbnail view of the image. If

the thumbnail view was the primary property, software applications such as browser could display it by default.

[0045] **Figure 5** is an exemplary service model in one embodiment of the invention. Each box in **Figure 5** represents a class or an interface. A dashed arrow indicates that the interface from which the arrow emanates can produce at run-time objects implementing the classes to which the arrow points. A content repository's implementation of the SPI is responsible for mapping operations on the content model to the particulars of a given content repository. *Repository* interface **500** represents a content repository and facilitates connecting to it. The *Repository* has a *connect()* method that returns an object of type *Ticket* **502** (or ticket) if a user is authenticated by the repository. In one embodiment, tickets are intended to be light-weight objects. As such, one or more may be created and possibly cached for each client/software application accessing a given repository.

[0046] A ticket can utilize a user's credentials to authorize a service. In one embodiment, a ticket can be the access point for the following service interfaces: *NodeOps* **508**, *ObjectClassOps* **506**, and *SearchOps* **510**. An application program can obtain objects that are compatible with these interfaces through the API *RepositoryManager* class. The *NodeOps* interface provides CRUD methods for nodes in the VCR. Nodes can be operated on based on their id or through their path in the node hierarchy. **Table 2** summarizes *NodeOp* class functionality exposed in the API.

***NodeOps* FUNCTIONALITY**
Update a given node's properties and property definitions.
Copy a given node to a new location in a given hierarchy along with all its descendants.
Create a new content node underneath a given parent.
Create a new hierarchy node underneath a given parent.
Perform a full cascade delete on a given node.
Retrieve all the nodes in a given node's path including itself.
Retrieve content node children for the given parent node.
Retrieve hierarchy node children for the given parent node.
Retrieve a node based on its ID.
Retrieve a node based on its path.

Retrieve the children nodes for the given hierarchy node.
Retrieve the parent nodes for the given hierarchy node.
Retrieve all the nodes with a given name.
Retrieve the Binary data for given node and property ids.
Moves a node to a new location in the hierarchy along with all its descendants.
Remove the ObjectClass from a given node.
Renames a given node and implicitly all of its descendants paths.
Get an iterator object which can be used to iterate over a hierarchy.

Table 2: *NodeOps* Functionality

[0047] **Figure 6** is an illustration of *NopeOps* service interaction in one embodiment of the invention. Application **600** utilizes a *NodeOps* object **602** provided by the API which in turn utilizes one or more *NodeOps* objects **606-610** provided by an SPI *Ticket*. Each repository **612-616** is represented by a *NodeOps* object. When the API *NodeOps* 602 receives a request to perform an action, it maps the request to one or more SPI *NodeOps* objects **606-610** which in turn fulfill the request using their associated repositories. In this way, applications and libraries utilizing the API see a the VCR rather than individual content repositories.

[0048] As with the *NodeOps* service, there is one SPI *ObjectClassOps* object per repository and a single API *ObjectClassOps* object. The API *ObjectClassOps* object maps requests to one or more SPI *ObjectClassOps* which in turn fulfill the requests using their respective repositories. Through this service, *ObjectClass* and *PropertyDefinition* objects can be operated on based on their id or through their path in the node hierarchy. **Table 3** summarizes *ObjectClassOps* class functionality exposed in the API.

ObjectClassOps FUNCTIONALITY
Create an *ObjectClass*, create *PropertyDefinition*(s) and associate them with the *ObjectClass*.
Add a given *PropertyDefinition* to an *ObjectClass*.

Delete an *ObjectClass*.
Delete a *PropertyDefinition*.
Retrieve an *ObjectClass* with a given id.
Retrieve all *ObjectClass*(es) available for all content repositories a given user is currently authenticated for.
Retrieve all of the *ObjectClass*(es) available for a given content repository.
Retreive a *BinaryValue* for the given *PropertyChoice*.
Retreive a *PropertyDefinition*.
Retrieve all *PropertyDefinitions* for the given *ObjectClass*.
Rename the given *ObjectClass*.
Updates the given *PropertyDefinition*.

Table 3: *ObjectClassOps* Functionality

[0049] As with the *NodeOps* and *ObjectClassOps* services, there is one SPI *SearchOps* object per repository and a single API *SearchOps* object. The API *SearchOps* object maps requests to one or more SPI *SearchOps* which in turn fulfill the requests using their respective repositories. Among other things, the *SearchOps* services allows applications and libraries to search for properties and/or values throughout the entire VCR. In one embodiment, searches can be conducted across all *Property*, *Value*, *BinaryValue*, *ObjectClass*, *PropertyChoice* and *PropertyDefinitions* objects in the VCR. Search expressions can include but are not limited to one or more logical expressions, Boolean operators, nested expressions, object names, function calls, mathematical functions, mathematical operators, string operators, image operators, and Structured Query Language (SQL). **Table 4** summarizes *SearchOps* class functionality exposed in the API.

SearchOps FUNCTIONALITY
Flushes all nodes inside a content cache.
Flushes a specified node from a content cache.
Performs a search with the given search expression.

Updates a content cache's attributes.
Updates a content cache's active state.
Updates a content cache's max entries.
Updates a content cache's time-to-live attribute.

Table 4: Exemplary *SearchOps* Functionality

[0050] **Figure 7** is an illustration of a VCR browser in one embodiment of the invention. A VCR browser **700** can include one or more tools built atop the API and has a graphical user interface (GUI). In one embodiment, the browser can be rendered using Microsoft Windows® (available from Microsoft, Corp.). In yet another embodiment, the browser can be implemented as a web portal. Browser window **700** includes a navigation pane **702** and a context-sensitive editor window **704**. The navigation pane displays a hierarchical representation of a VCR having one content repository ("BEA Repository") which itself has four hierarchy nodes ("HR", "Images", "Marketing", and "Products"). Selection of a hierarchy node can cause its children to be rendered beneath it in the navigation pane and cause an appropriate editor to be displayed in the editor window. Selection may be accomplished by any means, including but not limited to mouse or keyboard input, voice commands, physical gestures, etc. In this case, the VCR **706** is selected and a repository configuration editor is displayed in the editor window. The editor allows a user to change the configuration parameters (see **Table 1**) of the VCR. In one embodiment, configuration parameters are manipulated via Java Management Extensions (see **Figure 1**).

[0051] **Figure 8** is an illustration of a content editor in one embodiment of the invention. Navigation pane **802** is in "content" mode **812** such that it selectively filters out nodes that define only schemas. Content node **806** ("Laptop") has been selected. Node **806** is a child of hierarchy node "Products", which itself is a child of repository "BEA Repository". Selection of node **806** causes a corresponding content node editor to be rendered in editor window **804**. The editor displays the current values for the selected node. The content type **814** indicates that the schema for this node is named "product". In this example, the node has five properties: "Style", "Description", "Color", "SKU" and "Image". A user is allowed to change the value

associated with these properties and update the VCR (via the update button **808**), or remove the node from the VCR (via the remove button **810**).

[0052] **Figure 9** is an illustration of a schema editor in one embodiment of the invention. Navigation pane **902** is in "type" mode **910** such that it only displays nodes that have schemas but no content. Schema node **906** ("product") has been selected. Node **906** is a child of repository "BEA Repository". Selection of node **906** causes a corresponding schema editor to be rendered in editor window **904**. The editor displays the current schema for the selected node (e.g., derived from *ObjectClass, PropertyDefinition, PropertyChoice* objects). In this example, the node has five property definitions: "Style", "Description", "Color", "SKU" and "Image". For each property, the editor displays an indication of whether it is the primary property, its data type, its default value, and whether it is required. A property can be removed from a schema by selecting the property's delete button **912**. A property can be added by selecting the "add property" button **908**. A property's attributes can be changed by selecting its name **914** in the editor window or the navigation pane **906** (see **Figure 10**).

[0053] **Figure 10** is an illustration of a property editor in one embodiment of the invention. The schema named "product" is being edited. Schema properties definitions are listed beneath their schema name in the navigation pane **1002**. Schema property **1008** ("color") has been selected. The editor window **1004** displays the property's current attributes. The name of the attribute (e.g., "color"), whether the attribute is required or not, whether it is read-only, whether it is the primary property, its data type, default value(s), and whether the property is single/multiple restricted/unrestricted can be modified. Changes to the a property's attributes can be saved by selecting the update button **1006**.

[0054] **Figure 11** is an illustration of content mining system in one embodiment of the invention. Although this diagram depicts objects/processes as functionally separate, such depiction is merely for illustrative purposes. It will be apparent to those skilled in the art that the objects/processes portrayed in this figure can be arbitrarily combined or divided into separate software, firmware or hardware components. Furthermore, it will also be apparent to those skilled in the art that such objects/processes, regardless of how they are combined or divided, can execute on the same computing device or can be distributed among different computing devices connected by one or more networks.

[0055] Referring to **Figs. 1** and **11**, content mining system (CMS) **1100** can extract content from file systems and/or websites and transfer this content (or a reference/link to the content) to one or more repositories **108** via VCR **100**. In one embodiment, the CMS can traverse a file system and/or website and identify content therein in the form of files, HTML or XML documents, images, sounds, and/or any other kind of suitable content. This content can then be provided along with appropriate schema information to the VCR via the API **104** for inclusion in one or more repositories **108**. In one embodiment, content can be mapped to content nodes and directories can be mapped to hierarchy nodes. This can preserve the hierarchical relationship of content that arises from the organization of a file system or a website.

[0056] In one embodiment, one or more filter processes **1102** can be utilized. A filter process can analyze a particular kind of content and extract from it one or more properties. A property can include a name and an associated value. The following non-limiting example illustrates two properties:

```
name="author", content="John Smith"
name="description", content="Programmer"
```

[0057] The first property has a name of "author" and a value of "John Smith". The second property has a name of "description" and a value of "Programmer". Alternatively, properties can be specified more compactly in the form *name=value*, as in:

```
author = "John Smith"
description="Programmer"
```

[0058] In one embodiment, properties can incorporated into HTML documents as meta tags. By way of a non-limiting example, the following HTML code segment contains the same two properties as in the previous example:

```
<html>
      <head>
            <meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
            <title>Test Title</title>
            <meta name="author" content="John Smith">
            <meta name="description" content="Programmer">
      </head>
</html>
```

[0059] In another embodiment, a filter process can derive properties based on knowledge of the content type. By way of a non-limiting example, an image file (e.g.,

Joint Photographic Experts Group (JPG) file, Graphics Interchange Format (GIF) file, etc.), can be analyzed to determine an image's dimensions, resolution and other related information. This derived information can then be formatted into a set of properties.

[0060] In another embodiment, a filter process can supplement the properties associated with a given content wholly or partially by a supporting properties file. By way of a non-limiting example, the supporting properties file can be named with the same prefix as the filing containing the content, but with a different file extension. By way of a non-limiting example, an image file could be supplemented by a file including the following properties:

```
author="John Smith"
adTarget="engineers"
```

[0061] In one embodiment, a filter process can associate a schema with the properties. As discussed previously in reference to **Fig. 4** and elsewhere, schemas can be represented by an *ObjectClass* **408** object and zero or more *PropertyDefinition* **410** objects. For content mining purposes, a schema can be specified as another property. By way of a non-limiting example:

```
schema="advertisement"
author="John Smith"
adTarget="engineers"
```

[0062] In this example, an *ObjectClass* having the name "advertisement" and associated *PropertyDefinition* objects for "author" and "adTarget" is assumed to exist in the VCR. If the *ObjectClass* does not exist, it can be created dynamically by the CMS based on the properties associated with the content. In another embodiment, a schema can be specified in an XML document which can be accessed by a filter process in conjunction with content. Such a schema can be provided to the CMS which can in turn use it to dynamically create a schema in the VCR. In one embodiment, a filter process knows where to find schema information based on the type of content. For example, a filter process could search the current directory for a schema, a database, a website, a data structure or object, or other suitable location.

[0063] In one embodiment, there is the notion of a default schema. As the CMS is traversing a file system or website, it can select the most recently encountered schema (or some other schema) as the default schema for content that does not specify

one. By way of a non-limiting example, if the CMS is recursively traversing a directory structure in a depth-first fashion, then it will carry the default schema "down" the directory tree and associate it with content that lacks a schema property.

[0064] In one embodiment, filter processes can interact with the CMS via an API **1104** or some other suitable mechanism. The API **1104** can include services for allowing the CMS to direct a filter process to extract properties from the content (or provides references/links to the content properties). In one embodiment, a filter process can be an object accessible by the CMS. In another embodiment, a default filter process can be provided for content types that the CMS does not recognize.

[0065] **Figure 12** is an illustration of a content mining process in one embodiment of the invention. Although this figure depicts functional steps in a particular order for purposes of illustration, the process is not limited to any particular order or arrangement of steps. One skilled in the art will appreciate that the various steps portrayed in this figure could be omitted, rearranged, combined and/or adapted in various ways.

[0066] In step **1200**, a determination is made as to whether there is more content to "mine" or extract from a file system and/or website. By way of a non-limiting example, if the CMS recursively traverses a file system, this determination may evaluate to false once every directory in the file system has been visited. In that case, processing can complete. Otherwise, processing continues at steps **1202** and **1204**. Step **1202** associates a schema with the content using one of the approaches discussed above. Likewise, step **1208** extracts properties (or provides references/links to the properties) from the content using one of the previously discussed approaches. Steps **1202** and **1204** can be executed sequentially (in any order), concurrently, or in parallel with each other.

[0067] At step **1204**, a determination is made as to whether the schema identified in step **1202** already exists in the target repository. If not, it is created anew in the target repository in step **1206**. In another embodiment, if a schema by the same name but having a different definition exists in the target repository, it can be replaced by the schema identified in step **1202** or an error can be declared. At step **1210**, a determination is made as to whether the properties (or references/links to the properties) generated in step **1208** already exists in the target repository. If not, they are created anew in the target repository in step **1212**. Processing continues at step **1200** until there is no more content to process.

[0068] One embodiment may be implemented using a conventional general purpose or a specialized digital computer or microprocessor(s) programmed according to the teachings of the present disclosure, as will be apparent to those skilled in the computer art. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art. The invention may also be implemented by the preparation of integrated circuits or by interconnecting an appropriate network of conventional component circuits, as will be readily apparent to those skilled in the art.

[0069] One embodiment includes a computer program product which is a storage medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the features presented herein. The storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0070] Stored on any one of the computer readable medium (media), the present invention includes software for controlling both the hardware of the general purpose/specialized computer or microprocessor, and for enabling the computer or microprocessor to interact with a human user or other mechanism utilizing the results of the present invention. Such software may include, but is not limited to, device drivers, operating systems, execution environments/containers, and user applications.

[0071] The foregoing description of the preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. Embodiments were chosen and described in order to best describe the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention, the various embodiments and with various modifications that are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.